The 2010 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 18-22, 2010, Taipei, Taiwan

`Tiji`, a Generic Trajectory Generation Tool for Motion Planning and Control

Vivien Delsart and Thierry Fraichard

***Abstract*— Trajectory generation consists in computing a feasible trajectory between a start and a goal state-time, for a given robotic system. We presented in our previous works a trajectory generator called `Tiji`, geared towards complex dynamic systems subject to differential constraints. Moreover, it is able to handle a *final time constraint*, *ie* an interval of time during which the goal state must be reached, and to provide an admissible trajectory that ends close to the goal state-time if no solution exists to connect both states. This paper is a natural extension of these works, and presents several applications of our trajectory generator. Arbitrary robotic systems may be handled. Furthermore, the control-oriented nature of `Tiji` and its ability to handle a *final time constraint* makes it a useful tool to embed into various reactive approaches (trajectory tracking, obstacle avoidance). Simulation and experimental results illustrate the different systems and navigation approaches in which `Tiji` has been embedded.**

***Index Terms*— Trajectory generation; Differential constraints; Dynamic environments.**

I. Introduction

A. Motivation

The motion planning problem is a key question for an autonomous system to evolve in its environment. It has been largely addressed during the past forty years (see [1] for a review). To compute a motion, a robotic system must first consider its own dynamics and prevent any collision with the obstacles of its environment in the future. Motion safety may not be ensured if the system only tries to avoid collision at the current time, it must anticipate the obstacles' motion in the future.

This paper addresses the problem of trajectory generation, *ie* determining a feasible trajectory for a given robotic system(that respects the system's dynamics) between an initial and a final state-time. From the preliminary works of [2] to the recent methods used by the Carnegie Mellon University during the *Darpa Urban Challenge* ([3]), many trajectory generation methods have been proposed. While *primitive combination* [2], [4], [5], [6] approaches concatenate fixed geometric primitives to constitute a path up to the goal, *two-point boundary value problems* [7], [8], [9], [10], [11] try to find a high-degree specific type of curve connecting both start and goal states. Finally, *variational approaches* [12], [13], [14], [3] try to modify an initial parametric representation of a curve until both initial and goal states are connected and the constraints of the system are respected.

Among all these approaches, it is interesting to note that, in no circumstances, have people tried to compute a trajectory reaching the goal state at a specific *time instant* or during a fixed *time interval*. We presented in [15] a trajectory generation scheme called `Tiji` taking into account a *final time constraint* restricting the time at which a goal state may be reached. Many applications may be handled more easy with such an approach. A simple intersection problem requires taking into account a time interval fixed by the urban traffic to the robotic system to cross the intersection. Recent obstacle avoidance schemes [16], [17], [18] suppose a forecast model of the future of the environment to compute the next control to apply on the robotic system to anticipate the obstacles motion. From such a model of the future one may be interested to determine a trajectory reaching a temporary obstructed region during a collision-free time interval.

Multiple-robot coordination [19], [20] may require planning trajectories ending at fixed times too: If the different agents acting together cross their paths, a consensus algorithm may be used to decide priorities, nevertheless they may only be granted fixed time to obstruct a concurrent way without disturbing the other agents.

B. Contribution

The trajectory generator `Tiji`, based on a constraint optimization method, has been designed to handle the *final time constraint*, *ie* an interval of time during which the goal state must be reached. This constraint further increases the complexity of the trajectory generation problem since time now becomes an additional dimension to the problem. If it has not been well defined the goal state may be unreachable. Should this case appear, `Tiji` returns a trajectory that ends as close as possible to the goal state and guarantees its feasibility.

This paper is a natural extension of the works presented in [15] and presents `Tiji` as a generic tool for motion planning and control. We illustrate here different reactive motion determination approaches (computation of the motion to apply during the next time step) in which `Tiji` has been implemented. The control-oriented nature of `Tiji`, its efficiency (real-time computation) and above all, its ability to compute an alternative feasible trajectory when the goal state-time is unreachable, allow for example to perform obstacle avoidance in a trajectory deformation scheme, or to compute a control law in a trajectory tracking method. Furthermore, `Tiji` has been designed to handle arbitrary robotic systems. Results of the implementation of `Tiji` for several robotic systems and its integration in the applications cited above constitute the main content of this paper.

INRIA, CNRS-LIG & Grenoble University, France.

C. Outline of the Paper

The trajectory generator `Tiji` is briefly presented in Section II. The different robotic systems studied are presented in III and simulation results of the trajectory generation for such systems are given in Section IV. Several applications of our algorithm are presented in Section V. Conclusions are finally presented in Section VI.

II. Overview of the Approach

A. Notations and Definitions

Let $\mathcal{A}$ denote a robotic system operating in a workspace $\mathbf{W}$ ($\mathbb{R}^2$or $\mathbb{R}^3$). The dynamics of $\mathcal{A}$ are described by:

$$\dot{s} = f(s, \tilde{u}) \tag{1}$$

where $s \in \mathbf{S}$ is the state of $\mathcal{A}$, $\dot{s}$ its time derivative and $u \in \mathbf{U}$ a control. $\mathbf{S}$ and $\mathbf{U}$ respectively denote the state space and the control space of $\mathcal{A}$. Let $\tilde{u} : [0, t_f[\longrightarrow \mathbf{U}$ denote a control trajectory, *ie* a time-sequence of controls. Starting from an initial state s_0 (at time 0) and under the action of a control trajectory $\tilde{u}$, the state of $\mathcal{A}$ at time t is denoted by $\tilde{u}(s_0, t)$. A couple $\tilde{s} = (s_0, \tilde{u})$ defines a state trajectory for $\mathcal{A}$, *ie* a curve in $\mathbf{S} \times \mathbf{T}$ where $\mathbf{T}$ denotes the time dimension.

B. Trajectory Generation Problem

Given two states s_0 and s_g, trajectory generation consists in finding the control trajectory $\tilde{u}$ to apply from state s_0 in order to reach the goal state s_g. Let us note s_f the final state reached by a candidate trajectory $\tilde{u}$ applied from s_0 during a time t_f.

$$s_f = \tilde{u}(s_0, t_f) \tag{2}$$

The first condition the candidate trajectory $\tilde{u}$ must fulfill to be a solution of the trajectory generation problem is:

$$\mathbf{c}(s_0, \tilde{u}, t_f, s_g) = |s_g - s_f| = 0 \tag{3}$$

These constraint are named in the literature as the *state constraints*.

The control trajectory $\tilde{u}$ determined must nevertheless fulfill two other constraints: First, $\mathcal{A}$ is subject to a set of *motion constraints ie* bounds over its control and state parameters:

$$\mathbf{h}(s_0, \tilde{u}) \leq 0 \tag{4}$$

Second, we want to take into account a *final time constraint*, *ie* a range of time $[t_{min}; t_{max}]$ during which the trajectory may end. The final time t_f of the control trajectory $\tilde{u}$ must thus belong to this range of time:

$$t_{min} \leq t_f \leq t_{max} \tag{5}$$

C. Variational Trajectory Generation

We used a variational method to answer our trajectory generation problem. It consists in solving a minimization problem as follows :

$$\begin{array}{ll} \text{minimize :} & \mathbf{J}(s_0, \tilde{u}, t_f, s_g) = \sum_{j=1}^{n} \lambda^j ({s_g}^j - {s_f}^j)^2 \\ \text{subject to :} & \mathbf{h}(s_0, \tilde{u}) \leq 0 \\ & t_{min} \leq t_f \leq t_{max} \end{array} \tag{6}$$

where ${s_g}^j$ (resp. ${s_f}^j$) represents the j-th feature of the goal state (resp. final state) and λ^j its associated weight. The cost function $\mathbf{J}(s_0, \tilde{u}, t_f)$ represents thus a Mahalanobis distance between the final and the goal states. It is minimal when both states are equals. Bt applying iterative corrections on the input control, the cost function is minimized to help the final state of the trajectory to converge to the goal state while satisfying the motion and final time constraints.

Algorithm 1: `Tiji`

Input: $s_0, s_g, [t_{min}; t_{max}]$
Output: p, `success flag`

```
i = 0;
(p, t_f) = InitialGuess(s_0, s_g, [t_min; t_max]);
repeat
    step 1: Compute an admissible control trajectory ũ*_(p,t_f);
    step 2: Update the parameters p to converge to the goal state;
    i = i + 1;
until J(s_0, ũ, t_f, s_g) ≤ ε or i = i_max ;
return (p, t_f, s_f = s_g?);
```

A solution to perform the minimization process is proposed in [15]. Algorithm 1 recall the main stages of this process and additional details are provided below.

1) Parametrization of the control trajectory: To find a control trajectory $\tilde{u}$ that minimize $\mathbf{J}$, we first use a parametrization of its profiles to reduce the search space. We note then $\tilde{u}_{(p,t_f)}$ the parametrization of the control trajectory for a fixed vector of parameters $p = (p_1, \ldots, p_k)$ and a fixed final time t_f. Note that both p and t_f may be modified during the optimization process, however the *final time constraint* prevent t_f from taking a value outside $[t_{min}; t_{max}]$.

At the first step of the algorithm 1, an initial guess of parameters must be provided. Convergence to the goal greatly depends on the choice of the initial guess, it shall not be neglected. A look up table *ie* a recording of a sampling of the state space and the associated parameters to reach them may be a good option to compute them.

2) Computation of an admissible trajectory: A state trajectory $\tilde{s}_{(p,t_f)}(s_0)$ is then determined by integrating the dynamics of the system given in Eq. 2. Successive modifications of the parameters (p,t_f) during the optimization process may render $\tilde{s}_{(p,t_f)}(s_0)$ unfeasible ie. no guarantee is provided that the *motion constraints* given in Eq. 4 are fulfilled. Furthermore, the *final time constraint* may prevent the goal state from being reachable. If it appears, we would like to provide a trajectory that ends as close as possible to the goal state but that guarantee to be feasible.

A saturation of the control and state profiles is proceeded to ensure it (see [15] for details). It consists in determining intervals I_u and I_s where the control and state constraints are respectively overreached. Piecewise parametric profiles

are thus used to represent the feasible control trajectories $\tilde{u}^*_{(p,t_f)}(t, I_u, I_s)$ given by:

$$\tilde{u}^*_{(p,t_f)}(t, I_u, I_s) = \begin{cases} \mathbf{u}_{extl}(t) & \text{if } t \in I_u \\ 0 & \text{if } t \in I_s \\ \tilde{u}_{(p,t_f)}(t) & \text{otherwise} \end{cases} \quad (7)$$

where $\mathbf{u}_{extl}(t)$ is the extremal (minimal or maximal) control value that fulfill the *motion constraints* at time t.

3) Correction computation: Once a feasible trajectory has been computed, its final state s_f is easily determined as follows:

$$s_f = \tilde{s}^*_{(p,t_f)}(s_0, t_f) \quad (8)$$

If this final state s_f is enough close to the goal state s_g, *ie* if $\mathbf{J}(s_0, \tilde{u}, t_f, s_g) \leq \varepsilon$, where ε is a fixed threshold, the variational approach has converged, and a solution has been found. In the other case, we have to apply a correction over the set of parameters (p, t_f).

A local minimum of the cost function is found when its partial derivative $\left[\frac{\partial \mathbf{J}}{\partial (p,t_f)}\right]$ wrt. (p, t_f) are equals to 0. A root of $\left[\frac{\partial \mathbf{J}}{\partial (p,t_f)}\right]$ may thus be found by linearizing its expression as follows :

$$\left[\frac{\partial \mathbf{J}}{\partial (p,t_f)}\right] \simeq \left[\frac{\partial^2 \mathbf{J}}{\partial (p,t_f)^2}\right] \Delta(p, t_f) \quad (9)$$

where $\Delta(p, t_f)$ is the supposed error made over the set of parameters and $\left[\frac{\partial^2 \mathbf{J}}{\partial (p,t_f)^2}\right]$ are the 2nd order partial derivatives of the cost function wrt. parameters. To minimize the cost function $\mathbf{J}$, a correction over the set of parameters may thus be computed:

$$cor_{(p,t_f)} = -\tau \left[\frac{\partial^2 \mathbf{J}}{\partial (p,t_f)^2}\right]^{-1} \left[\frac{\partial \mathbf{J}}{\partial (p,t_f)}\right] \quad (10)$$

where $\tau \in [0; 1]$ is the gain of the correction. The inverted matrix of the 2nd order partial derivatives represents then the direction of the correction applied, and $\tau \left[\frac{\partial \mathbf{J}}{\partial (p,t_f)}\right]$ represents the step length of the newton descent method.

From that point, a new trajectory can be computed. The process is repeated until convergence or failure (after a fixed number of steps if for instance, the algorithm is blocked in a local minimum).

III. Case Studies

We introduce briefly in the section two robotic system for which the trajectory generator `Tiji` has been implemented. Samples of trajectories generated for each system are presented in section (IV).

A. Differential Drive System

To illustrate the trajectory generation process `Tiji`, it has been first applied to the case of a planar differential drive system $\mathcal{A}_{dd}$. A state of a differential drive system $\mathcal{A}_{dd}$ is defined by a 5-tuple $s = (x, y, \theta, \omega_L, \omega_R)$ where (x, y) are the coordinates of the center of the wheels, θ is the main orientation of $\mathcal{A}_{dd}$ and ω_L (resp. ω_R) is the angular velocity of the left (resp. right) wheel. A control of $\mathcal{A}_{dd}$ is defined by the couple $u = (\eta_L, \eta_R)$ where η_L (resp. η_R) is the angular acceleration of the left (resp. right) wheel.
So knowing the velocities of the wheels, the main linear and angular velocities v and ω of $\mathcal{A}_{dd}$ are given by :

$$v(t) = \frac{\omega_R(t) + \omega_L(t)}{2} \qquad \omega(t) = \frac{\omega_R(t) - \omega_L(t)}{2 * \mathbf{b}} \quad (11)$$

where $\mathbf{b}$ is the length between the center of the differential drive system and the wheels.
Then the motion of $\mathcal{A}_{dd}$ is governed by the following differential equations:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\omega}_L \\ \dot{\omega}_R \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \\ \eta_L \\ \eta_R \end{pmatrix} \quad (12)$$

We will consider then the followings constraints over the differential drive system :

$$(\omega_L, \omega_R) \in [-\omega_{max}; \omega_{max}]^2, (\eta_L, \eta_R) \in [-\eta_{max}; \eta_{max}]^2 \quad (13)$$

B. Car-Like System

A state of a car-like system $\mathcal{A}_{cl}$ is defined by a 5-tuple $s = (x, y, \theta, \phi, v)$ where (x, y) are the coordinates of the rear wheel, θ is the main orientation of $\mathcal{A}$, ϕ is the orientation of the front wheels (steering angle), and v is the linear velocity of the rear wheel. A control of $\mathcal{A}_{cl}$ is defined by the couple $u = (a, \zeta)$ where a is the rear wheel linear acceleration and ζ the steering velocity. The motion of $\mathcal{A}$ is governed by the following differential equations:

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\phi} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ v\tan(\phi)/L \\ \zeta \\ a \end{pmatrix} \quad (14)$$

where L is the wheelbase of $\mathcal{A}$ and:

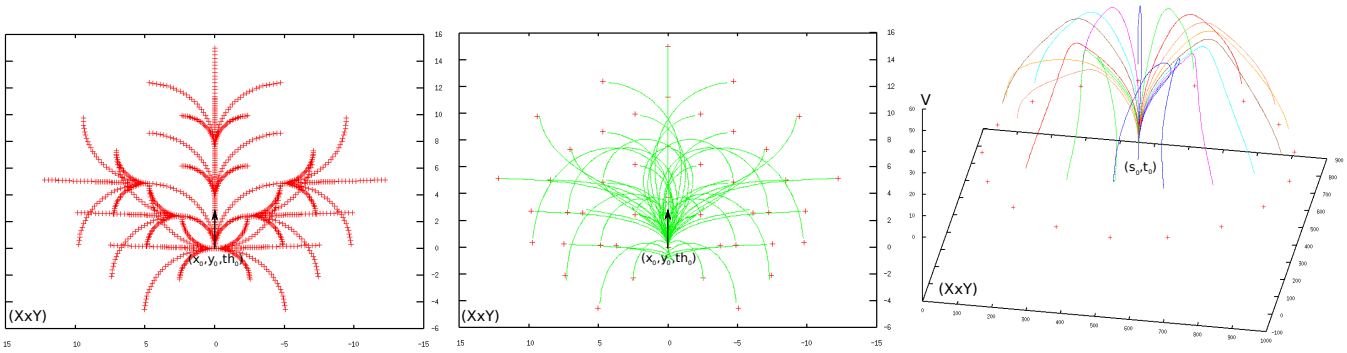
$$v \in [0, v_{\max}], |\phi| \leq \phi_{\max}, |a| \leq a_{\max} \text{ and } |\zeta| \leq \zeta_{\max} \quad (15)$$

IV. Simulation Results

`Tiji` has been implemented in C++ and tested on a desktop computer (Core-i7@3.4GHz, 6GB RAM, Linux OS). It has been evaluated in different scenarios featuring both reachable and unreachable goal state-times. The maximum number of iterations was heuristically set to 20.

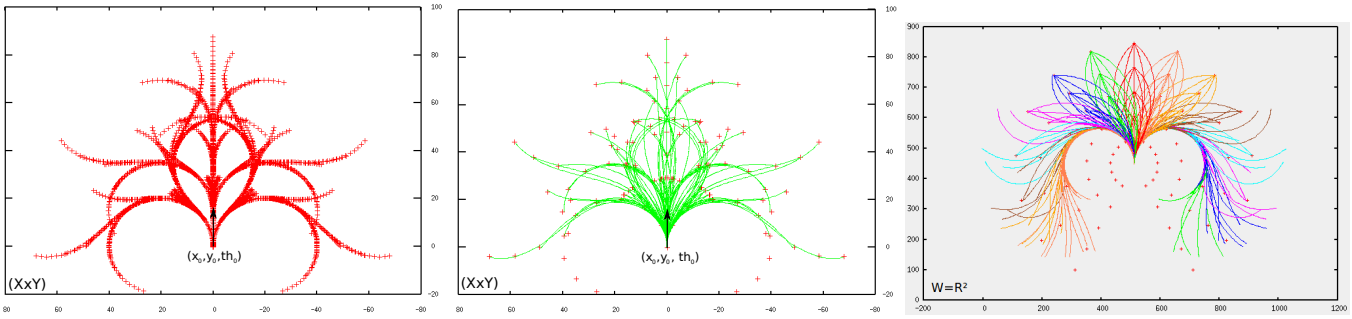
A. Reachable Goal State-times

To evaluate performances of the trajectory generator `Tiji`, sampling trees have been implemented for each robotic system. Sampling trees consist in determining the topology of the reachable space for a given robotic system in computing a tree of feasible trajectories from a sampling of available controls at each time step. All target states defined



(a) Sampling tree ($x \times y$ view) (b) Trajectories generated ($x \times y$ view) (c) Unreachable final states ($x \times y \times v$ view)

Fig. 1: Sampling tree describing a subset of the available trajectories for a differential drive system, trajectories generated by `Tiji` to reach its end state-times (leaves), and examples of generated trajectories when the goal state-times (crosses) are not reachable.



(a) Sampling tree ($x \times y$ view) (b) Trajectories generated ($x \times y$ view) (c) Unreachable final states ($x \times y$ view)

Fig. 2: Sampling tree describing a subset of the available trajectories for a car-like system, trajectories generated by `Tiji` to reach its end state-times (leaves) and examples of generated trajectories when the goal state-times (crosses) are not reachable.

Robotic system	Differential Drive System	Car-like Vehicle
number of evaluated states-times	327680	393216
success rate (%)	98.80	96.52
average required steps (in case of success)	3.23	7.08
average time by trajectory (ms)	2.84	5.12

TABLE I: Performances of the trajectory generation for each case study in the case where all goal state-times are reachable.

by the leaves of the tree are consequently reachable. The trajectory generator `Tiji` has been used to try to reach all final states (leaves) of the sampling trees computed for each robotic system defined in Section III. Figures 1 and 2 present respectively the sampling trees and generated trajectories for a differential drive and a car-like system.

From a complexity point of view, the trajectory generation algorithm depends on the number of required steps to converge to the goal state. Table I sum up the average number of required steps and the resulting computation time in the three different cases study. Thanks to look-up tables of initial guesses of parameters, quite good success rates of convergence to the goal state-times have been obtained, while computational times are very low.

B. Unreachable Goal State-times

As `Tiji` is aimed at computing feasible trajectories that end "as close as possible" to the goal state-time when it is not reachable, we propose some results here to show the relevance of the alternative trajectories computed. Figures 1c and 2c present respectively samples of the trajectories computed for a differential drive and a car-like systems.

To conclude on the relevance of the alternative trajectories, we computed an evaluation coefficient μ for each state trajectory $\tilde{s}$ starting from s_0 that does not reach the goal state s_g at a fixed final time t_f: We propose to compare the distance from the goal state s_g to the closest state s_{RS} of the reachable set $\mathcal{R}(s_0, t_f)$ with the distance $\|s_g - s_f\|$. We compute thus the evaluation coefficient μ defining below to conclude on the quality of the alternative trajectories computed:

$$\mu = \frac{\|s_g - s_f\|}{\|s_g - s_{RS}\|} \tag{16}$$

Robotic system	Differential Drive System	Car-like Vehicle
number of evaluated state-times	10000	10000
average time by trajectory (ms)	13.56	10.41
average evaluation coefficient μ	1.14	1.15

TABLE II: Performances of the trajectory generation for each case study in the case where the goal state-times are not reachable.

Note than a metric is needed to compute distances between these states. This coefficient tend to 1 if the final state s_f found is exactly the best solution wrt. the metric used.

The performances obtained for unreachable cases are summarized in table II. The computation times remains really low while the average error between final and goal states (described by the average evaluation coefficients) seems satisfactory.

V. Applications

The method presented in section II is highly constrained and aimed at solving complex motion planning problems. However, fixing or freeing the state and final time constraints allows to use the trajectory generator `Tiji` in a large selection of motion planning and control applications. This section presents two applications of `Tiji` : a trajectory deformation method and a trajectory tracking process.

A. Trajectory Deformation

Trajectory deformation is a technique that was introduced to generate robot motion wherein a trajectory, that has been computed beforehand, is continuously deformed on-line in response to moving and unforeseen obstacles. We proposed in [21] a trajectory deformation approach named `Teddy`. Its principle is simple: a complete motion to the goal is computed first using a priori information. It is then passed on to the robotic system for execution. During the course of the execution, the still-to-be-executed part of the motion is continuously deformed in response to sensor information acquired on-line, thus accounting for the incompleteness and inaccuracies of the a priori world model.

The deformed trajectory is sampled as a sequence of nodes in the state-time space $\mathbf{S} \times \mathbf{T}$. Two types of forces are thus applied on each node: external forces (imposed by the obstacles) and internal forces (to maintain the connectivity of the trajectory). External forces move each node away from a priori model of the future behavior of obstacles. During this process, the connectivity of the trajectory may be lost: there is no guarantee that a feasible trajectory still exists between each couple of state-times (nodes). Our trajectory generator `Tiji` is aimed to restore the connectivity of the trajectory in such a case. Consider three successive state-times (s_-, t_-), (s, t) and (s_+, t_+). A feasible trajectory is computed by `Tiji` between (s_-, t_-) and (s_+, t_+) if it exists. In the other case, `Tiji` provide a trajectory from (s_-, t_-) that ends as close as possible to (s_+, t_+). The intermediate state-time (s, t) is thus brought back to the middle state-time (given at time $(t_+ - t_-)/2$) of the generated trajectory. Obstacle avoidance is thus performed while the connectivity of the trajectory is kept.

The need of the *final time constraint* is well illustrated here: the guarantee of the temporal consistency of the trajectory deformed, and maintaining its connectivity impose to fix the time or at least an interval of time in which each intermediate node may be moved.

Figure 3 depicts an example of the trajectory deformation process for a diffenrential drive.

B. Trajectory Tracking

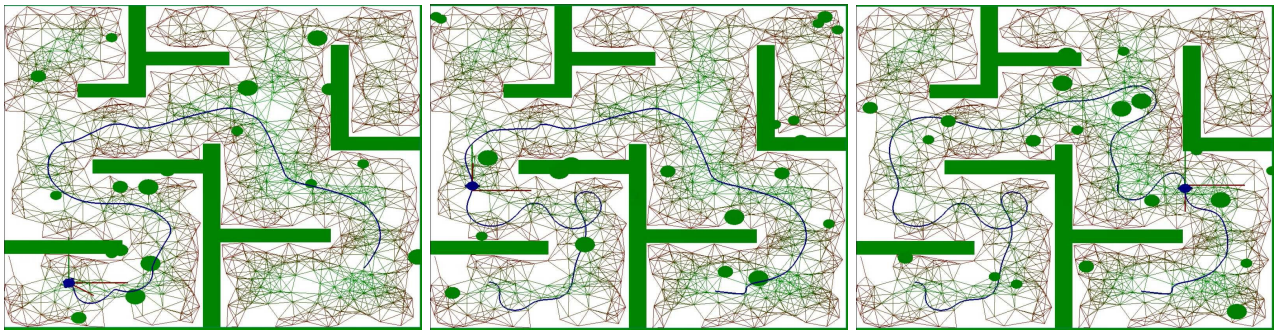
Trajectory tracking is the problem of reaching and following a trajectory of the state-time space $\mathbf{S} \times \mathbf{T}$ (*ie* a geometric path with an associated timing law) starting from a given initial configuration. A robotic system with perfect sensor and control models does not require a trajectory tracking model. It is aimed to compensate for the possible derivation from an initial trajectory to follow due to sensor and actuator errors. It consists then in trying to determine at each time instant the input control to apply on the robotic system, given the current robot's state perception.

The main problems of such an approach is first, to choose at each time instant a state-time of the trajectory followed to reach, and second to find the appropriate control to move as close as possible to it. By choosing a fixed look-ahead time to select the state-time to reach and using `Tiji` at each time step from the current position of the robot to reach the select goal with a fixed final time constraint, our approach limits the instability of the trajectory tracking process. Figure 4 presents an application of our trajectory tracker using `Tiji` on an automated wheelchair. The look-ahead time is fixed here to 0.5 second. Fast computation of the trajectory generation process allows to compensate for errors on the control applied and on the localization. The resulting derivation from the reference trajectory never exceed 0.25 meters whatever the speed used to follow the trajectory.

VI. Conclusion and Future Work

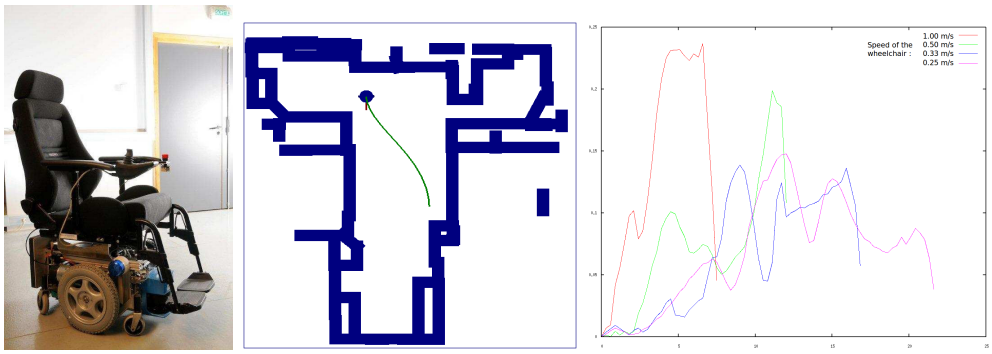
The paper has presented `Tiji`, a new trajectory generation scheme that can be used to *efficiently* compute *feasible* trajectories for system with complex dynamics. Besides, it can handle a *final time constraint*, *ie* reaching a goal at a prescribed time instant. When the goal is unreachable, it returns a trajectory ending as close as possible to the goal.

`Tiji` can handle arbitrary robotic systems, and may be used in many reactive navigation applications. Its efficiency has been illustrated by simulation results in a trajectory deformation process to keep the connectivity of the deformed trajectory, and by experimental results on an automated wheelchair in a trajectory tracking process to used determine the control to apply at each time as input of the robotic system.



(a) $t = 0$ (b) $t = 10$ (c) $t = 25$

Fig. 3: Differential drive evolving amongst static and moving obstacles : The different figures represent the deformed trajectory at different time steps ($x \times y$ view).



(a) Automated wheelchair (b) Reference trajectory (c) Derivation of the position along the trajectory (in meters) while following the reference trajectory at different speeds

Fig. 4: Illustration of the trajectory tracking process using `Tiji` on an automated wheelchair. Fast computation of the trajectory generation process allows to follow efficiently a reference trajectory at different speeds.

REFERENCES

[1] S. M. Lavalle, *Planning Algorithms*. Cambridge Univ. Press, 2006.

[2] L. Dubins, "On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents," *American Journal of Mathematics*, vol. 79, 1957.

[3] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *Int. Journal of Robotics Research*, 2007.

[4] J. Reeds and L. Shepp, "Optimal paths for a car that goes forward and backward," *Pacific J. Math.*, vol. 145, 1990.

[5] Y. Kanayama and N. Miyake, "Trajectory generation for mobile robots," in *In Proc. of the Int. Symp. on Robotics Research*, 1985.

[6] T. Fraichard and A. Scheuer, "From reeds and shepp to continuous curvature paths," *Transactions on Robotics*, vol. 20, 2004.

[7] K. Komoriya and K. Tanie, "Trajectory design and control of a wheel-type mobile robot using b-spline curve," in *Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots ans Systems*, 1989.

[8] A. Takahashi, T. Hongo, and Y. Ninomiya, "Local path planning and control for AGV in positioning," in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1989.

[9] Y. Kanayama and B. I. Hartman, "Smooth local path planning for autonomous vehicles," in *IEEE Int. Conf. on Robotics and Automation*, 1989.

[10] A. Piazzi and C. Guarino Lo Bianco, "Quintic G^2-splines for trajectory planning of autonomous vehicles," in *Proc. of the IEEE Intelligent Vehicles Symp.*, 2000.

[11] F. Lamiraux and J.-P. Laumond, "Smooth motion planning for car-like vehicle," *IEEE Trans. on Robotics and Automation*, vol. 17, 2001.

[12] H. Delingette, M. Hébert, and K. Ikeuchi, "Trajectory generation with curvature constraint based on energy minimization," in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 1991.

[13] P. Gallina and A. Gasparetto, "A technique to analytically formulate and to solve the 2-dimensional constrained trajectory planning problem for a mobile robot," *Journal of Intelligent and Robotic Systems*, 2000.

[14] A. Kelly and B. Nagy, "Reactive nonholonomic trajectory generation via parametric optimal control," in *The International Journal of Robotic Research*, vol. 22, Jul.-Aug. 2003.

[15] V. Delsart, T. Fraichard, and L. Martinez-Gomez, "Real-time trajectory generation for car-like vehicle navigating dynamic environments," in *IEEE Int. Conf. on Robotics and Automation*, 2009.

[16] L. Martinez-Gomez and T. Fraichard, "Collision avoidance in dynamic environments: an ics-based solution and its comparative evaluation," in *Proc. of the Intl Conf. on Robotics & Automation*, May 2009.

[17] M. Seder and I. Petrovic, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles," in *Proc. of the Intl Conf. on Robotics & Automation*, April 2007.

[18] F. Large, C. Laugier, and Z. Shiller, "Navigation among moving obstacles using the nlvo : Principles and applications to intelligent vehicles," *Autonomous Robots Journal*, vol. 19, no. 2, September 2005.

[19] Y. Li, K. Gupta, and S. Payandeh, "Motion planning of multiple agents in virtual environments using coordination graphs," in *IEEE Int. Conf. on Robotics and Automation*, Barcelona, Spain, april 2005.

[20] A. S. Oikonomopoulos, S. G. Loizou, and K. J. Kyriakopoulos, "Coordination of multiple non-holonomic agents with input constraints," in *IEEE Int. Conf. on Robotics and Automation*, may 2009.

[21] V. Delsart and T. Fraichard, "Navigating dynamic environments using trajectory deformation," in *IEEE-RSJ Int. Conf. on Intelligent Robots and Systems*, 2008.